APPLICATION UNDER UNITED STATES PATENT LAWS

Atty. Dkt. No. PW 281100
(M#)

Invention: SYSTEM, METHOD, AND APPARATUS FOR EARLY CULLING

Inventor (s): MOREIN, Stephen L.

00909

Pillsbury Winthrop LLP

This is a:

- ☐ Provisional Application
- ☒ Regular Utility Application
- ☐ Continuing Application
  - ☐ The contents of the parent are incorporated by reference
- ☐ PCT National Phase Application
- ☐ Design Application
- ☐ Reissue Application
- ☐ Plant Application
- ☐ Substitute Specification
  Sub. Spec Filed _____
  in App. No. _____ / _____
- ☐ Marked up Specification re
  Sub. Spec. filed _____
  In App. No _____ / _____

SPECIFICATION

PATENT APPLICATION
DOCKET NO. 010007BM

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

FILING OF A UNITED STATES PATENT APPLICATION

SYSTEM, METHOD, AND APPARATUS FOR EARLY CULLING

INVENTOR:

Stephen L. Morein
10 Magazine St.
Cambridge, MA 01752

ATTORNEYS OF RECORD

William P. Atkins
Jeffrey D. Karceski
Kerry Hartman

Pillsbury Winthrop LLP
1600 Tysons Boulevard
McLean, VA 22102
(703) 905-2000

SYSTEM, METHOD, AND APPARATUS FOR EARLY CULLING

Related Applications

[0001] This application claims benefit of U.S. Provisional Patent Application No. 60/330,676, filed October 29, 2001, entitled "SYSTEM, METHOD, AND APPARATUS FOR EARLY CULLING".

Field of the Invention

[0002] This invention relates to video graphics processing.

Background Information

[0003] Graphics rendering is an important part of many representational and interactive applications for computers. In three-dimensional (or '3D') graphics rendering, an image of a 3D rendering space is presented on a display frame as if the space is being viewed through a two-dimensional display plane. As shown in FIGURE 1, the display frame 10 is an array of individual picture elements (or 'pixels') 30. Each pixel represents a sample of the display plane at a specified location and has a color value that corresponds to the color of the rendering space as viewed through the display plane at that location. For consumer applications, typical sizes for a display frame include 640 × 480, 800 × 600, 1024 × 768, 1152 × 864, 1280 × 1024, and 1600 × 1200 pixels. Computer displays and other high-resolution display devices, such as high-definition televisions (HDTVs), projectors, printers, and the like, present the pixel array to a viewer. The pixels are closely spaced, and the viewer's visual system performs a filtering of the individual pixels to form a composite image. If an image is properly partitioned into pixels that are sufficiently close together, the viewer perceives the displayed array as a virtually continuous image.

[0004] In a surface rendering scheme, three-dimensional 'wire frame' models of objects in the rendering space are constructed using graphics primitives (e.g. triangles or other elemental polygons). Each primitive is defined by a set of vertices that have values

relating to location (e.g. in an XYZ coordinate space), quality (e.g. color and/or texture), and/or lighting (e.g. direction of surface normal). As the positions of the vertices with respect to the display plane include the spatial dimension of depth, or distance from the viewer (also referred to as the Z-dimension), objects may be drawn in front of or behind one another in overlapping fashion.

[0005] Some or all of the rendering of the object models into pixels for display may be performed in software. Alternatively, the sets of vertices may be presented to rendering hardware, either directly by the software application or via an application programming interface (API) such as the Direct3D component of the DirectX API suite (Microsoft Corp., Redmond, WA) or the OpenGL API (Silicon Graphics, Inc., Mountain View, CA).

[0006] FIGURE 2 shows a block diagram for a 3D video graphics architecture. Such an architecture may be implemented in a video graphics card for use in a personal computer. Alternatively, at least a portion of the architecture may be implemented on a chip package that is integrated into a computer mainboard.

[0007] Rasterizer 120 receives a set of graphics primitives for each of the objects to be rendered in a scene. In this example, rasterizer 120 receives each primitive as a triangle having vertices as described above. Rasterizer 120 may receive the primitives from a software application such as a game, simulator, or other imaging application. Alternatively, as shown in FIGURE 3, rasterizer 120 may receive the primitives from a transform and lighting engine 110, which may perform coordinate transform and/or lighting operations on the primitives. Rasterizer 120 converts the primitives into fragments, each fragment being a bundle of values to update a particular pixel of display frame 10 as represented in a frame buffer 170. For example, a fragment may include a value for each of the components in the colorspace (e.g. RGB, HSV, YCbCr) and an opacity or 'alpha' value. The fragment values may also indicate a corresponding location in the rendering space as well as corresponding locations in one or more texture, lighting, or environment maps.

[0008] Rasterizer 120 forwards the fragments to a pixel pipeline 130, where their values may be modified by processing-intensive operations such as smoothing, blending, and dithering and/or access-intensive operations such as texture and bump mapping. The particular operations that pixel pipeline 130 performs on a fragment may depend upon the

current configuration of the pipeline (e.g. as defined by the current values of a set of pipeline state variables).

[0009] Render backend 160, which receives the fragments from pixel pipeline 130, includes a fragment culler 150 and a pixel combiner 140. Fragment culler 150 discards fragments according to the results of one or more culling tests. For example, fragment culler 150 may perform an occlusion test (or 'Z-test') by comparing a Z value of the fragment to a value of Z buffer 180 that corresponds to the same pixel. If a fragment is not discarded by the fragment culler, the corresponding pixel in frame buffer 170 may be updated (e.g. blended or replaced) by pixel combiner 140 according to the fragment's color value. Other values of a surviving fragment may be used to update corresponding locations in other display buffers as well: for example, the fragment's z-value may be used to update a corresponding location of Z buffer 180. When all of the objects to be rendered have been rasterized and incorporated into the display buffers, the contents of frame buffer 170 are modulated onto a display signal (not shown).

[0010] In order to enhance the appearance of a generated image, the pipeline may be configured to perform several operations on a fragment. For example, the color and/or alpha values of a fragment may be altered with reference to one or more effect maps (e.g. texture, bump, and light maps). Such operations may be costly in terms of processor cycles and/or memory bandwidth. If the fragment culler subsequently determines that the fragment is occluded and discards it, the resources expended on that fragment in the pixel pipeline (e.g. in terms of processor cycles and/or memory bus usage) have been wasted. It is desirable to reduce such waste.

SUMMARY

[0011] A method of graphics processing according to one embodiment of the invention includes determining a non-depth conditional status and an occlusion status of a fragment. Determining a non-depth conditional status of a fragment includes determining whether incorporation of a value (e.g. a color value) of the fragment into the pixel is conditional on a non-depth criterion. For example, this task may include determining a current configuration of a pixel pipeline (e.g. determining a value of at least one state variable) or determining whether a non-depth fragment test (such as an alpha test) is enabled.

Determining an occlusion status of the fragment may include comparing another value (e.g. a Z value) of the fragment to a value of an entry of a scratchpad, where the entry is mapped to the pixel. Determining a conditional status and determining an occlusion status may occur in either order, and methods according to further embodiments may also include comparing a value (e.g. a Z value) of the fragment to a representative Z value (e.g. corresponding to a location value of the fragment).

[0012] A graphics architecture according to an embodiment of the invention includes an early culler and a scratchpad. The scratchpad is configured and arranged to store a value of an entry mapped to a pixel. The early culler is configured and arranged to receive a fragment corresponding to the pixel, to compare a first value (e.g. a Z value) of the fragment to the value of the entry, and to determine whether incorporation of a second value (e.g. a color value) of the fragment into the pixel is conditional on a non-depth criterion. Architectures according to other embodiments of the invention may also include a pixel pipeline configured and arranged to receive the fragment from the early culler.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] FIGURE 1 shows a division of a display frame into pixels;

[0014] FIGURE 2 shows a block diagram of a 3D graphics architecture;

[0015] FIGURE 3 shows a block diagram of another 3D graphics architecture;

[0016] FIGURE 4 shows a block diagram of a 3D graphics architecture according to an embodiment of the invention;

[0017] FIGURE 5 shows a division of a display frame into pixel blocks;

[0018] FIGURE 6 shows a flow chart of a method according to an embodiment of the invention;

[0019] FIGURE 7 shows a flow chart of a method according to another embodiment of the invention;

[0020] FIGURE 8 shows a flow chart of a method according to a further embodiment of the invention;

[0021] FIGURE 9 shows a flow chart of a method according to a further embodiment of the invention;

[0022] FIGURE 10 shows a flow chart of a method according to a further embodiment of the invention;

[0023] FIGURE 11 shows a flow chart of a method according to a further embodiment of the invention;

[0024] FIGURE 12 shows a flow chart of a method according to a further embodiment of the invention; and

[0025] FIGURE 13 shows a flow chart of a method according to a further embodiment of the invention.

DETAILED DESCRIPTION

[0026] Depth information is important to the 3D rendering process, as an object that is entirely opaque will occlude those portions of other objects that lie behind it with respect to the viewpoint. Because values of occluded fragments will not be entered into the display buffers (or will be overwritten before display), effort expended in rendering these fragments may be wasted. In some cases, processing overhead and/or memory bus usage may be considerably reduced by avoiding operations associated with rendering occluded fragments.

[0027] In a 3D graphics architecture, it may be desirable to identify fragments that will be discarded and cull them before they reach the pixel pipeline. Specifically, it may be desirable to identify and cull fragments that will be occluded. However, it may not be feasible to determine at the pipeline entrance whether a fragment ultimately will be occluded. One reason for this uncertainty is that the survival of potentially occluding fragments that are currently being processed within the architecture may still be unknown.

[0028] In addition to a Z-test, a graphics architecture may perform one or more other tests to determine whether a fragment will be incorporated into a frame buffer or will be

discarded. These tests may include a pixel ownership test, a scissor test, an alpha test, a stencil test, and/or a chroma-key test:

- A pixel ownership test determines whether the current context owns (e.g. may modify) the pixel that corresponds to the fragment.
- A scissor test determines whether the fragment lies within a specified portion of the display frame.
- An alpha test determines whether an alpha value of the fragment has a specified relation to a reference alpha value.
- A stencil test determines whether a value of a location of the stencil buffer that corresponds to the fragment has a specified relation to a reference value.
- A chroma-key test determines whether a color value of the fragment has a specified relation to a reference value.

In order to configure and enable these tests, a graphics architecture may include a set of state variables whose values may be modified (e.g. via an associated API). For example, these values may include reference values for the tests as well as control values that toggle specified tests between enabled and disabled modes.

[0029] Depending on the outcome of such non-depth fragment tests, a fragment may survive to be incorporated into the corresponding pixel, or it may be discarded. Until such tests are completed, it is unknown whether this fragment may occlude others.

[0030] As shown in FIGURE 4, a graphics architecture according to an embodiment of the invention includes an early culler 100 that compares Z values of fragments to Z values stored in a scratchpad 200. Fragments that fail the comparison (i.e. fragments identified as occluded) are culled. Fragments that pass the comparison are passed to the pixel pipeline, and their Z values may also be stored to the scratchpad. (As discussed herein, fragments that are far away from the viewpoint have higher Z values than fragments that are close, although other orientations of the Z-axis are also possible.) Early culler 100 also performs a task of determining a non-depth conditional status of a fragment as described herein.

[0031] FIGURE 5 shows how display frame 10 may be divided into pixel blocks 20. Although FIGURE 5 depicts pixel blocks 20 as square and of equal size, the division may be performed in any other manner that is efficient with respect to a particular application (e.g. in light of such factors as instruction set architecture, datapath width, and storage or tiling format). For example, blocks that are rectangular and/or interleaved may be used, or blocks of differing shapes and sizes may be distributed throughout a display frame. The same division scheme, or a different one, may be used for different rendering operations (e.g. for operations relating to color values as opposed to operations relating to Z values).

[0032] In an exemplary implementation, scratchpad 200 is organized into lines, each line being mapped to a predetermined region (e.g. a block) of display frame 10, and each entry in a line being mapped to a corresponding pixel within the predetermined region. As with a cache, the mapping of scratchpad 200 to display frame 10 may be direct-mapped, fully associative, or (n-way) set-associative. In this implementation, each entry of scratchpad 200 has a valid flag, and each line of scratchpad 200 also has a valid flag.

[0033] In a system that includes a stencil buffer, a portion of each Z value may be reserved for use in stenciling operations such as shadow processing. For example, eight bits of a 32-bit Z-value, or one bit of a 16-bit value, may be reserved for one or more stencil values. It may be desirable to omit the stencil portions when storing such fragment Z values to the entries of scratchpad 200.

[0034] Early culler 100 and scratchpad 200 may be implemented in a single chip package integrated on a computer mainboard. Early culler 100 and scratchpad 200 may also be implemented in a video graphics card for use in a personal computer. Alternatively, scratchpad 200 may reside at least in part in system memory. Although it is possible for early culler 100 to be implemented as a dedicated unit, or as one or more processes executing on a dedicated array of logic elements, it may be more efficient for early culler 100 to be implemented as one or more processes executing on an array that also performs some or all of the other processing tasks of a 3D graphics architecture (e.g. tasks of transform and lighting engine 110, rasterizer 120, and/or pixel pipeline 130).

[0035] FIGURE 6 shows a flow chart for a method according to an embodiment of the invention. Task T120 receives a fragment (e.g. from rasterizer 120) and determines a non-depth conditional status of the fragment. That is, task T120 determines whether

incorporation of a value of the fragment (e.g. a color value) into the corresponding pixel is conditional on a criterion other than the fragment's Z value. For example, task T120 may determine that the pipeline is configured to perform an alpha test on the fragment, such that a failure of the test would prevent the fragment from being incorporated into the corresponding pixel. If incorporation of the fragment into the corresponding pixel is non-depth conditional, then task T130 passes the fragment to the pipeline and the method terminates.

[0036] In one implementation, task T120 simply determines whether any of a predetermined set of non-depth fragment tests are enabled. In an OpenGL application, for example, task T120 may call the function glIsEnabled with arguments GL_ALPHA_TEST, GL_SCISSOR_TEST, and/or GL_STENCIL_TEST and test the results. In another implementation, task T120 also determines whether such a test as configured may actually cull any fragments. For example, while an alpha test may be enabled, it may also be configured to pass any fragment having an alpha value greater than or equal to zero. In such a case, although the test is enabled, it will not actually cull any fragments (assuming that the alpha values are constrained to be nonnegative).

[0037] Task T120 may also determine whether the pipeline is scheduled for a possible modification of a Z value of the fragment (e.g. as may occur in certain texture-based operations). In this case, if incorporation of the fragment into the corresponding pixel is non-depth conditional or if a modification of the fragment's Z value is possible, then task T130 passes the fragment to the pipeline and the method terminates.

[0038] If task T120 determines that the fragment is robust (i.e. not conditional on the result of a non-depth fragment test), then task T140 queries the scratchpad to determine whether any entry is currently mapped to a pixel that corresponds to the fragment's location value. If a scratchpad miss occurs (i.e. no entry is currently mapped to that location), task T150 determines whether any invalid scratchpad line may be mapped to a region (e.g. a block) that includes the pixel. If such a scratchpad line is found, task T160 maps the line accordingly, stores the Z value of the fragment to the appropriate entry, and marks the line and the entry as valid (the other entries of the line remain marked as invalid until fragment Z values are stored to them).

[0039] If task T150 fails to find an invalid scratchpad line that may be mapped to include the pixel, task T170 applies a predetermined replacement policy to select a scratchpad

line for replacement. The selected scratchpad line is remapped, the fragment is stored to the appropriate entry, and the other entries of the line are invalidated. Task T130 passes the fragment to the pipeline and the method terminates.

[0040] Task T170 may include any appropriate cache replacement policy. For example, the replacement policy may be based at least in part on a least-recently-used (LRU) criterion. It may also be desirable for the replacement policy to account for the age of the scratchpad line. In a further implementation, the contents of the selected line are written to a miss cache or a victim cache, and task T140 checks the miss cache or victim cache before deciding that a scratchpad miss has occurred (for further discussion of miss caching and victim caching, see Norman P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," WRL Technical Note TN-14, Digital Equipment Corporation, Palo Alto, CA, 1990). In an implementation where the scratchpad is direct-mapped, no selection of a scratchpad line is necessary in task T170.

[0041] If a scratchpad hit occurs in task T140, task T145 determines whether the corresponding entry is valid. If the entry is not valid, task T200 stores the Z value of the fragment to the entry and sets the valid flag. If the entry is valid, Z test task T180 compares its value to the Z value of the fragment. If the test succeeds (e.g. the Z value of the fragment is less than the value of the entry), then task T200 replaces the value of the entry with the Z value of the fragment (in this case, the valid flag is already set). The method passes the fragment to the pipeline and terminates in task T130. If the Z test of task T180 fails, the fragment is culled and the method terminates in task T190.

[0042] One technique for reducing Z-buffer traffic is to implement a representative Z value memory (also called a 'hierarchical Z buffer' or 'HRZ buffer') as described in co-pending U.S. Patent Applications Nos. 09/140,930, entitled "METHOD AND APPARATUS FOR PROCESSING OBJECT ELEMENTS THAT ARE BEING RENDERED," and 09/141,218, entitled "METHOD AND APPARATUS FOR EFFICIENT CLEARING OF MEMORY," both filed August 27, 1998 and assigned to the assignee of the present application. This technique supports a conservative culling operation that uses less storage space and memory bandwidth than pixel-resolution Z buffering.

[0043] A hierarchical Z buffer that is updated only upon a Z cache miss may not contain the most current representative Z information. For example, changes that occur

within a block may not be reflected in the HRZ buffer until the corresponding cache line is replaced. One effect of this latency is that a HRZ buffer test may pass fragments that a test using current representative Z information would have culled. For example, a HRZ buffer test may fail to cull fragments of a self-occluding object.

[0044] In contrast, changes that occur within a block may be reflected more quickly in a scratchpad as described herein, and a method using such a scratchpad may provide improved culling performance. FIGURE 7 shows a flow chart of a method according to an embodiment of the invention that may be practiced in a system including a hierarchical Z buffer. In this method, task T135 compares the Z value of the robust fragment to the representative Z value that corresponds to the fragment's location value. If the test fails (i.e. the fragment is identified as occluded), the method terminates in task T190 and the fragment is culled. Otherwise, the method proceeds with task T140 as described above.

[0045] Although a method according to an embodiment of the invention may be practiced without such capability, a method as shown in FIGURE 7 also includes updating the hierarchical Z buffer from valid scratchpad lines in tasks T210–T230 as described below. In this case, each line of scratchpad 200 corresponds to at least the same block of pixels as a representative Z value of the hierarchical Z buffer.

[0046] After the Z value of the fragment has been stored to the scratchpad line in task T200, task T210 determines whether all of the entries in the line are valid. If any entries are invalid, the method terminates in task T130 and the fragment is passed to the pipeline. Otherwise, task T220 identifies the backmost Z value ('BMZ value') among the entries of the line and compares it to the corresponding representative Z value. If the BMZ value is no closer to the display plane than the representative Z value, the method terminates in task T130 and the fragment is passed to the pipeline. Otherwise, task T230 updates the HRZ buffer by replacing the corresponding representative Z value with the BMZ value. In a further implementation, task T230 also marks the line as invalid (i.e. available for the storage of values of fragments corresponding to other blocks).

[0047] FIGURES 8 and 9 show flowcharts of methods according to embodiments of the invention in which fragments may be tested for culling without regard to their conditional status, but only robust fragments are used to update the scratchpad. In these cases, conditional test tasks T120a,b are performed only when the contents of the scratchpad are

about to be modified. This feature allows scratchpad-based culling of a fragment to be performed without any conditionality testing. FIGURE 9 shows a flowchart of a method in which HRZ-based culling of a fragment may also be performed without any conditionality testing.

[0048] FIGURES 10 and 11 show flowcharts of methods according to further embodiments of the invention wherein the entries of scratchpad 200 are initialized to a maximum Z value (i.e. in task T260). In other implementations, the entries of scratchpad 200 are initialized to the backmost Z value of the scene, of the Z buffer, or of the HRZ buffer. Alternatively, the entries of each line may be initialized to the representative Z value (from a hierarchical Z buffer) corresponding to that line. In such cases, the scratchpad may be implemented without using valid flags for entries. If the entire scratchpad is initialized, valid flags for lines may also be omitted. FIGURE 11 shows a flowchart according to a further implementation in which fragments may be tested for culling without regard to their conditional status.

[0049] In methods according to additional embodiments of the invention, tasks of scratchpad querying (as in task T140), conditional status testing (as in task T120), and HRZ buffer testing (as in task T135) may be performed in various sequences (e.g. according to efficiencies calculated or estimated with respect to the particular application or architecture) and with or without scratchpad entry initialization. FIGURES 12 and 13 show flowcharts of methods according to further embodiments of the invention that include HRZ buffer testing and wherein the entries of scratchpad 200 are initialized to a maximum Z value (or to another initial value as described above). A method as shown in FIGURE 13 also includes the feature (e.g. as discussed above with reference to FIGURES 8 and 9) wherein fragments may be tested for culling without regard to their conditional status.

[0050] The foregoing presentation of the described embodiments is provided to enable any person skilled in the art to make or use the present invention. Various modifications to these embodiments are possible, and the generic principles presented herein may be applied to other embodiments as well. For example, an embodiment of the invention may be implemented in part or in whole as a hard-wired circuit or as a circuit configuration fabricated into a video graphics integrated circuit or field-programmable gate array. Likewise, an embodiment of the invention may be implemented in part or in whole as a firmware program loaded or fabricated into non-volatile storage (such as read-only memory

or flash memory) as machine-readable code, such code being instructions executable by an array of logic elements such as a microprocessor or other digital signal processing unit. Further, an embodiment of the invention may be implemented in part or in whole as a software program loaded as machine-readable code from or into a data storage medium such as a magnetic, optical, magnetooptical, or phase-change disk or disk drive; a semiconductor memory; or a printed bar code.

[0051] Additionally, an embodiment of the invention may be applied to the early culling of fragments on non-depth bases as well: for example, according to opacity, color, or window or stencil membership. Thus, the present invention is not intended to be limited to the embodiments shown above but rather is to be accorded the widest scope consistent with the principles and novel features disclosed in any fashion herein.